CSCI1300: Final Project

Checkpoint - Due: July 5, 2020 at 11:59 pm (MT)
Interview Grading Required

Final Submission - Due: July 19, 2020 at 11:59 pm (MT)
Interview Grading Required

The final project for this course entails writing a larger, more complex program than any of the problem sets. You may choose to do any of the following options as your project. If you decide to choose your own adventure, you must write up a brief (about two paragraphs) proposal and get approval from the instructor.

1. Hangman

Write a C++ program for the game "Hangman" (https://en.wikipedia.org/wiki/Hangman_(game)).

The general **rules** are:

1. There are two players: one who thinks of a word (Player 1) and another who guesses the word (Player 2).
2. The game starts with a series of empty slots, one for each letter. For example: "apple" would be represented by "_ _ _ _ _".
3. On each turn, Player 2 guesses a letter. If the letter belongs to the word, Player 1 adds it to the corresponding position. For example: in "apple", if a player guesses p, we have "_ p p _ _".
4. If player 2 guesses a letter which does not belong to the word, player 1 draws one component of the hangman diagram (see "Example game").
5. At any time during the game, player 2 can guess the entire word (instead of just guessing a single character). If the guess is correct, the game is over. If it is incorrect, player 1 adds one component to the hangman diagram.
6. The game is over when either player 2 guesses the word (character by character or the entire word at once) or when the hangman diagram is completed.
7. You need to create a file which contains a list of words to be guessed. When your game starts, you pick a word randomly from this list which the player needs to guess.

At each turn, you need to ask the player whether they wish to guess the entire word or a single character.

After each guess, you need to display:

1. Whether the guess belonged to the word or not.
2. Characters guessed so far in the word. For instance, if ‘p’ was guessed in ‘apple’, display “_ p p _ _”.
3. Current state of the hangman diagram

You may make alternative design choices with respect to the hangman diagram such as keeping a “attempts left” count if that will make things easier to implement.

Apart from the requirements above:

- You **must** implement a Computer mode. In this mode, the computer will read a random word from a plaintext file, with a corresponding clue (so that the user can guess the word), present the same to the user and ask the user to guess the word.
  - In fact, you may choose to not implement the 2-player version and instead have only a Player vs Computer mode, where the player must guess.

- You should also keep track of how much time the user took to complete the attempt and have a leaderboard saved to a file where the shortest time ranks the highest.

2. Minesweeper

Write a C++ program for the game “Minesweeper” (https://en.wikipedia.org/wiki/Minesweeper_(video_game)).

The game works as follows:

1. The player would elect to “open” a cell on a 2D grid of cells. If the cell contains a bomb and the user opens it, the game is over. Otherwise the cell would contain a number indicating how many bombs are contained in the cells adjacent to itself (up to 8 for any given cell).

2. The player attempts to use the information revealed by each cell to flag cells where they believe a bomb exists.

3. The player wins the game when all cells containing a bomb have been flagged and all non-bomb cells have been opened.

You need to create a file which would populate the 2D grid for the game. It should indicate which cells contain bombs and the numbers corresponding to each non-bomb cell. The format for this input file is left to the student's discretion.

You also need to have an option to undo the flagging of a cell so that if a user makes a mistake while flagging any particular cell they do not have to start over.

The minimum size for the grid should be 6x6.

You should have a collection of such input files so that we know that your program does not depend on any specific configuration of the 2D grid in order to work properly. You should vary both the configuration of bombs and the size of the grid.

At each turn, you should:

1. Display the current state of the 2D grid.
2. Ask the user for a command: 'open', 'flag' or 'undo'
3. Ask the user for a location on the grid to apply the above command: x, y. For example : 0, 0 would correspond to the cell in the upper left corner.

Apart from the requirements above, you should also keep track of how much time the user took to complete the game and have a leaderboard saved to a file where the shortest time ranks the highest.

3. Classes Catalog

Write a C++ program to allow interaction with the classes catalog at a University. Take a look at classes.colorado.edu for inspiration – but your project need not be this extensive. The program must include the following features:

1. You should maintain a list of students and their details (name, studentID, password, codes of courses they have completed and *optionally* the corresponding grade) on a file. Store details in such a way that they can be imported while running the program. Plaintext files are fine here – we will ignore the security concerns in this course.

2. You will also need to maintain a list of courses and their respective details (course code, course title, instructor, pre-requisite course codes) in a

separate file. Store details in such a way that they can be imported while running the program. Again, plaintext files are fine.

3. Allow a student to login using their student ID and password. Load their profile and academic records into their program upon successful authentication.

4. Students should be able to enroll in the courses they select if they have completed at least 1 course in the given course’s prerequisites. Optionally, you may enforce a grade criteria as well.

When a user logs in, the program should list the relevant options (continuously until quitting) as follows: If the user is a registrar, the options allowed for registrars should be printed. If the user is a student, the options allowed for students should be printed.

NOTE: Please note that this is a relatively advanced project, so if you want to tweak the features (minor reductions, not major) to your convenience, that is fine with me.

4. Twitter

Write a program that is a bare-bones version of Twitter on the terminal. You must include the following features:

1. You should maintain a list of users and their personal details on a file. Store details in such a way that they can be imported while running the program. Plaintext files are fine here.

2. Let the user tweet a message. Make sure that the message length is less than 140/280 characters.

3. Allow users to export their own tweets to a plaintext file.

4. Allow users to follow other users, they must be valid users (can be accomplished by maintaining a vector or array of users).

When a user logs in, the program should display all their tweets and the tweets of the users they are following and give them the option to compose another, to (un)follow someone, or to quit.

NOTE: Please note that this is also a relatively advanced project, so if you want to tweak the features (minor reductions, not major) to your convenience, that is fine with me.

5. Choose Your Own Adventure

If you want to work on your own idea, write up a paragraph or two with a project proposal and email it to me (**sanskar.katiyar@colorado.edu**).

I will either approve your idea or ask you to meet me to discuss your proposal if it is not detailed enough. You must get your idea approved before starting your own project. Once approved, you may submit the approved proposal on the Project Checkpoint item.

You may also discuss prospective ideas with you TA before drafting the proposal.

Notes

Kindly note that these project ideas list only general requirements and unlike the problem sets, do not enforce any strict specifications. Your task is to make conscious design choices that will allow you to manage your codebase and project features in a relatively efficient manner.

The primary objective of the project is to give you control over these design choices and help you see the advantages/disadvantages of certain approaches when you are maintaining a relatively large codebase.

You are always welcome to consult any of the staff members during their office hours or specialized design meetings (prior appointments may be required).

If your project requires some sample data of sample files, you can generate it here: https://www.generatedata.com/. You may also request your TA for some sample files.

Project Checkpoint

You must submit a checkpoint document by July 5, 2020 on Canvas. You will also be required to attend an interview. Not attending an interview will result in a loss of **all** points despite a submission.

The document must contain the responses to the following questions:

1. Which project topic you would be working on?
2. List the prototypes of these functions and classes (if applicable) that you plan to use to solve the problem(s) at hand. Additionally, try to mention a general approach you will adopt to implement the function, does not need to be a complete algorithm or pseudocode.

A model submission might look like this:

```
int linear_search(int arr[], int size, int search_for)
{
  1. start a loop at index 0, go till end of array
        a. check if search_for is equal to arr[index]
        b. if it is: then return index
  2. if not found, return -1.
}
```

Kindly note that these function prototypes need not be correct or complete now. We just want to ensure that you are headed in the right direction and have a general idea on how to proceed. This would also help you ask clarifying or guiding questions from your interviewer.

You may submit a CPP, TXT or a PDF file with these function prototypes.

After submitting the said file, attend the interview. In the interview, ask any clarifying questions that you may have regarding your approach and seek references from the interviewer to help you get going.

Course's late submission policy applies and not attending an interview will result in a loss of **all** points despite a submission.

Final Submission

The final submission for the project must be submitted on Canvas by July 19, 2020.

The final deliverables for the project include:

1. Source code files (.cpp), Header files (.h) – required to compile and execute the program. Should be neatly organized in a ZIP file.
2. A brief report

The format for the report as well as the grading rubric will be available on the "Final Project" item on Canvas in due course.

Course's late submission policy applies. There will be a 10% deduction per late hour, i.e., if the due date is 11:59pm then submission between 12:00am – 00:59am will incur a 10% penalty and so forth.

There will be **zero-tolerance for plagiarism**. Your submission may be subject to additional plagiarism screening.